Express Mail No. EV 907 292 943 US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Application of: Olsen et al.	Confirmation No. 5312
Serial No.: 10/046,907	Art Unit: 2161
Filed: January 17, 2002	Examiner: Te Y. Chen
For: METHOD AND SYSTEM FOR STORING AND PROCESSING HIGH-FREQUENCY DATA	Attorney Docket No. 060967-0004-US

PRE-APPEAL BRIEF REQUEST FOR REVIEW

Mail Stop AF
Commissioner for Patent
P. O. Box 1450
Alexandria, VA 22313-1450

Sir:

In the final Office Action of March 2, 2006, claims 44-52, 55-66 and 84 were

rejected. The remaining claims have been withdrawn or canceled. A Notice of Appeal from

the rejection of claims 44-52, 55-66 and 84 is being filed simultaneously herewith.

Applicant's claims on appeal are directed to a system for processing time-stamped

data from one or more time series. As stated in paragraph 0029 of applicants' specification, a

time series is "a set of data points sorted in order of increasing time." Further, as set forth in

the claims, the time series is time-stamped. Time-stamping is described at paragraph 0037

and shown on the left-hand side of Fig. 1.

Applicants' system is directed in particular to the processing of financial time series

data. As noted at paragraph 0004 of the specification, such financial time series data tends to

be irregularly spaced as is also shown in the time-stamps of Fig. 1. As also noted in

paragraph 0004 such data also tends to be voluminous and not readily stored on conventional

computers.

To overcome these problems, applicants have devised a language and a system to process time-stamped, time series data on a data flow basis. An overview of this system is set forth at paragraphs 0157 through 0164. An example of the system is set forth in Fig. 9 and described in paragraphs 0221 through 0226 and the table of C++ code incorporated in paragraph 0221.

The exemplary system comprises several processing blocks. The Orla Read Ascii block reads Ascii data, the Orla Project block extracts a Bid price, the Orla EMA block computes an exponential moving average (EMA) and the Orla Print block generates an output. As set forth at paragraphs 0200 through 0204, each block may be thought of as a small processor or procedure and each block has input and/or output ports by which it is connected to other blocks in a network. The blocks are the processing modules of paragraph (a) of claim 44. Examples of various blocks are set forth at paragraphs 0210 through 0220.

The blocks are interconnected through ports as more fully described in paragraphs 0205 through 0209. The connection between an output port of one block and an input port of the next block is established by a binding operation described more fully at paragraphs 0209, 0223 and 0224. These elements constitute the connections for linking modules as recited in paragraph (b) of claim 44.

Finally, the system is managed by a network scheduler described at paragraph 0163. The system is activated by the statement "net.run ( )" as described at paragraphs 0225 and 0226.

The claims were rejected under 35 U.S.C. 102(e) as anticipated by Langseth et al. (U.S. Patent No. 6,741,980). Langseth describes a personal intelligence network that delivers personalized information and transactional data from a database to individuals via e-mail, phone, PDAs or the like. As best shown in Fig. 2A, the system includes a variety of channels of information 40 that provide input to a data distribution system 42. The data distribution

system outputs content either directly or through a variety of affiliates 202 using a variety of communication resources 22 such as the Internet, wireless communication and telephony. Fig. 2B provides more detail on the system.

A wide variety of content is contemplated for distribution by the Langseth system including sports information, business news, weather, travel information, financial information and a news channel.

At page 7 of the Office Action, the Examiner directs the applicants' attention in particular to the financial information described at Col. 3, lines 26-29 and that shown in Fig. 13. Col. 3, lines 26-29 describe a service that may be called "Market Update" that "sends an email to subscribers every day at 5 p.m. with a summary of the market results for the day." Fig. 13 appears to be similar. It is captioned "Daily Market Summary" and appears to show the closing prices, dollar change, high, low, volume and percent change for a group of stocks and two charts, one covering six days and the other seven months. Fig. 13 is described at Col. 6, lines 27-28 as a "facsimile output" and at Col. 22, lines 25-31 as a facsimile of a detailed chart.

Also in Langseth is a description at Col. 10, lines 9-51 of various types of services that might be provided over its financial channel. These services include moving averages, P/E ratios, earnings and a host of other financial data that can only be appreciated by reviewing the listing in Column 10.

With respect to the disclosure at Col. 3, lines 26-29, little information is given as to what is provided. It is merely characterized as "a summary of the market results." This could be in almost any format. No suggestion is given in this description that this information constitutes time series data or that such time series data is time stamped. Moreover, Col. 3, lines 26-29 merely describe the output of the financial service and do not describe what is processed by the financial service. Thus, this material provides no

suggestion of a system for processing time-stamped, time series data as claimed by applicants.

Similarly, with respect to Fig. 13 which the Examiner describes as "an example of the claimed time series data," Fig. 13 merely shows a table of closing prices for various stocks and two charts neither of which is explained. The set of closing prices does not constitute a time series because the closing prices are not sorted in the order of increasing time. As for the charts, they are not explained. But most important of all, the material of Fig. 13 is described as an output of Langseth's invention. This output does not teach or suggest what is provided to Langseth's system for processing and therefore does not teach or suggest applicants' system for processing time-stamped, time series data.

In the absence of any disclosure in Langseth of the input of time-stamped, time series data, there is no suggestion in Langseth of the system claimed in claim 44 which includes one or more processing modules for processing time-stamped, time series data. Moreover, there is no suggestion of the claimed connections for such modules or of a sub-system for activating such modules.

Accordingly, it is respectfully submitted that claim 44 is patentable.

Dependent claims 45-52, 55-66 and 84 are believed patentable for the same reason claim 44 is patentable. In addition, dependent claim 45 and claims 46-50 which are dependent thereon are believed patentable for the additional reason that they specify that the system of claim 44 further includes a type system comprising one or more types and a relation among them.

The type system is described at paragraphs 0197 to 0199, 0205 to 0209, and 0280 to 0383. Data is required to belong to a specific data-type and the processing blocks or modules specify what types of data they accept. As noted at paragraph 0295, a grammar describes all valid types in the type system.

Applicants' usage of the terms "type" and "type system" is intended to be consistent with the usage of these terms in computer science. See, for example, attached pages 12-16 of A.V. Aho and J.D. Ullman, Foundations of Computer Science, (Computer Science Press 1992), one of the leading texts in computer science.

While Langseth does use the word "type" at Col. 7, line 13, it is used in a general description of organizing information by relationships such as "subject matter, date, type, etc."; and nothing indicates that he is using the term in a technical fashion as applicants are using it. Accordingly, it is respectfully submitted that Langseth does not suggest a system for processing time-stamped, time series data that further includes a type system as recited in applicants' claim 45.

In view of the forgoing remarks, the claims in this application are believe to be in condition for allowance. Such action is respectfully requested.

Respectfully submitted,

Date: September 5, 2006

Francis E. Morris
Reg. No. 24,615
MORGAN, LEWIS & BOCKIUS LLP
Customer No. 009629
212-309-6632

Foundations of
Computer
Science

Alfred V. Aho
Jeffrey D. Ullman

Printed in the United States of America

BEST AVAILABLE COPY

reason is that until we compute the carry-out of the rightmost place, we cannot compute z_1 or the carry-out of the second place. Until we compute the carry-out of the second place, we cannot compute z_2 or the carry-out of the third place, and so on. Thus, the time taken by the circuit is the length of the numbers being added — 32 in our case — multiplied by the time needed by a one-bit adder.

One might suspect that the need to "ripple" the carry through each of the one-bit adders, in turn, is inherent in the definition of addition. Thus, it may come as a surprise to the reader that computers have a much faster way of adding numbers. We shall cover such an improved algorithm for addition when we discuss the design of circuits in Chapter 13. ✦

EXERCISES

1.3.1: Explain the difference between the static and dynamic aspects of a data model.

1.3.2: Describe the data model of your favorite video game. Distinguish between static and dynamic aspects of the model. *Hint:* The static parts are not just the parts of the game board that do not move. For example, in Pac Man, the static part includes not only the map, but the "power pills," "monsters," and so on.

1.3.3: Describe the data model of your favorite text editor.

1.3.4: Describe the data model of a spreadsheet program.

✦✦ 1.4   The Pascal Data Model

In this section we shall summarize the data model used by the programming language Pascal. We begin with the static part of the model, which in a programming language is often called the *type system*, because the static aspect of the model is really the collection of types that data may have. We then discuss the dynamics of the Pascal data model, that is, the operations that may be performed on data.

Type system

The Pascal Type System

The basic principle under which Pascal and most other programming languages deal with data is that each program has access to "boxes." Each box has a type, such as integer or the type CELL defined in Example 1.1. We may store in a box any value of the correct type for that box. We often refer to the values that can be stored in boxes as *data objects*.

Data object

We may also name boxes. In general, a *name* for a box is any expression that denotes that box. Often, we think of the names of boxes as the variables of the program, but that is not quite right. For example, if x is a variable local to a recursive procedure P, then there may be many boxes named x, each associated with a separate call to P. Then the true name of such a box is a combination of x and the particular call to P.

Name

As another example, the built-in function new of Pascal can be applied to a pointer variable, say p. When we execute new(p), p is made to point to a new box, of the proper type for something pointed to by p. For instance, if p is a pointer to integers, then the box is of type integer. The only name for this box is p^.

In Pascal, there is an infinite set of types, any of which could be the associated with a particular box. These types, and the rules by which the constructed, form the *type system* of Pascal. The type system contains eleme types such as integers, and a collection of type-formation rules with which w construct progressively more complex types from types we already know. In S 2.6, we shall discuss recursive definitions in general, of which the definition Pascal type system is an example. In recursive definitions, we start with a in this case the elementary types of Pascal, and continue with an *induction* where we apply formation rules as many times as we like, to construct all objects in which we are interested.

BASIS. For Pascal, the elementary types are

1.   Integers

2.   Real numbers

3.   Characters

4.   Booleans (TRUE and FALSE)

5.   Special finite sets of elements, which are either

   a.   *Range types* of integers, such as 10..20, or of characters, such as 'a'. or

   b.   *Enumerated types*, such as

   (Happy, Bashful, Grumpy, Sleepy, Sleazy, Hairy, Filthy)

INDUCTION. The type-formation rules assume that we already have some t which could be elementary types or other types that we have already constr using these rules. The definition of a type T can also include uses of T itself, dir or indirectly. For example, the definition of CELL in Section 1.1 used a field of LIST, where LIST = ^ CELL. In Pascal, we create new types using the follo rules:

1.   *Array types.* We can form an array whose elements are of some arbitrary t say T. The array is indexed by some finite set I, as defined by basis rule above. Then

   array [I] of T

   is an array type. For example, if our index type is 1..100 and the type 7 array elements is the type CELL discussed earlier, then

   array [1..100] of CELL

   is an array type.

2.   *Record-structure types.* We can form a record whose fields are arbitrary ty either basic or defined. Different fields can have different types, but each must have elements of a single type. The rule for forming record structure If $T_1, T_2, \ldots, T_n$ are arbitrary types and $F_1, F_2, \ldots, F_n$ are field names, the

record
$\quad F_1 : T_1 ;$
$\quad F_2 : T_2 ;$
$\qquad \cdots$
$\quad F_n : T_n$
end

defines a type with n fields. The ith field has name F_i and value of type T_i, for $i = 1, 2, \ldots, n$.

3. *Pointer types.* If T is any type, then $\hat{}\ T$ denotes the type "pointer to an object of type T." Notice that if p names a box of type $\hat{}\ T$, then the value in box p is a pointer, often drawn as an arrow, rather than an object of type T itself. When we learn about the architecture of a computer in Chapter 4, we shall see that what really appears in the box named p is the address, or location, at which a certain object of type T is stored in the computer. Figure 1.10 illustrates the conventional view of a pointer-valued variable.

4. *File types.* If T is any type, then `file of` T defines the type that is a file of objects of type T.

5. *Set types.* If T is a type that denotes a "small" set of elements, such as a range type with a sufficiently small range (the actual limit on the range depends on the particular Pascal compiler used), then `set of` T defines a type that is a set of objects of type T.
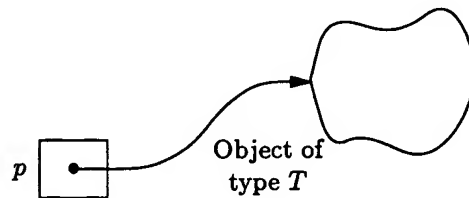

Fig. 1.10. Variable p is of type $\hat{}\ T$.

✦ **Example 1.4.** Consider the four type definitions in Fig. 1.11. In the conventional view of data in Pascal, an object of type TYPE1 is an array with 10 slots, each holding an integer, as suggested in Fig. 1.12(a). Likewise, objects of type TYPE2 are pointers to such arrays, as in Fig. 1.12(b). Record objects, like those of TYPE3, are visualized as in Fig. 1.12(c), with a slot for each field; note that the name of the field (e.g., field1) does not actually appear with the value of the field. Finally, objects of TYPE4 would have five slots, each of which holds an object of TYPE3, a structure we suggest in Fig. 1.12(d). ✦

✦ **Example 1.5.** Example 1.4 is typical of how we construct objects of a given type, provided that the type is not defined in terms of itself. If a type is defined in terms of itself, objects of that type may still make sense. A case in point is the

```
TYPE1 = array [1..10] of integer;

TYPE2 = ^ TYPE1;

TYPE3 = record
    field1: integer;
    field2: TYPE2
end;

TYPE4 = array [0..4] of TYPE3;
```
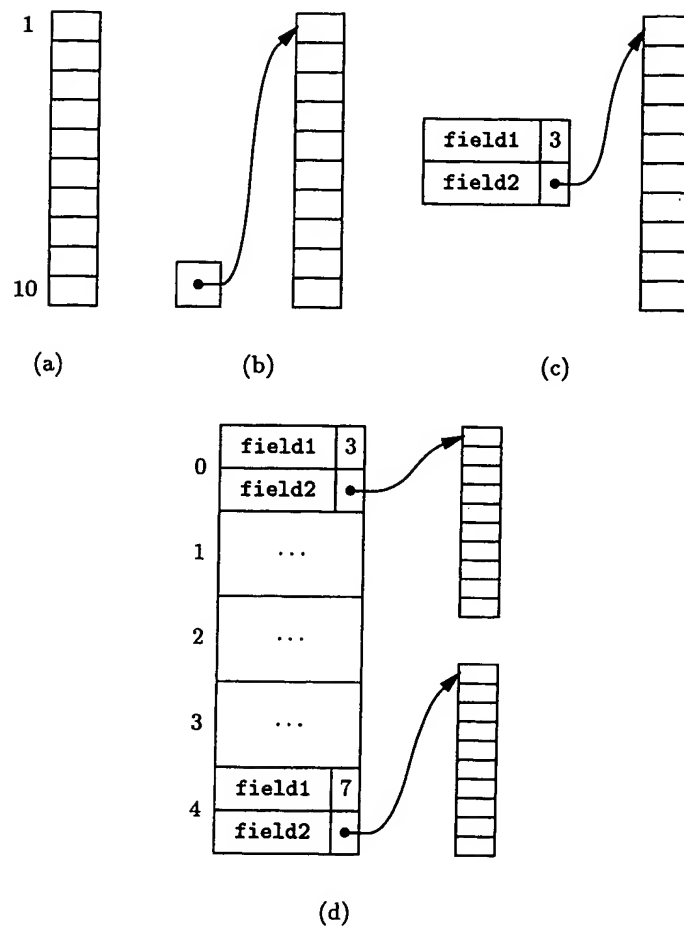
Fig. 1.11. Some Pascal type declarations.

(a)   (b)   (c)

(d)

Fig. 1.12. Visualization of type declarations in Fig. 1.11.

Types, Names, Variables, and Identifiers

A number of terms associated with data objects have different meanings but are ea to confuse. First, a type describes a "shape" for data objects; we have illustrat the Pascal type system, or collection of possible types, in this section, but oth programming languages may have other type systems. In Pascal, types are defin using the keyword type, as in

    type T = <type descriptor>

Type descriptor

The *type descriptor* is an expression that tells us the shape of objects of the type such as array [1..10] of ^ integer, that is, an array of 10 pointers to intege

A type definition for type T does not actually create any objects of that typ An object of type T is created by a declaration of the form

    var x: T

Here, x is an *identifier*, or "variable name," associated with some procedure P (a special case, P could be the whole program). When P is called, a box who name is "the x associated with this call to P" is created. Recall that a "nam is any expression that can be used to refer to a box. As mentioned in the te there can be many boxes each of whose name involves the identifier x, since P m be recursive. There may even be other procedures that also have used identifier to name one of their variables. Moreover, names are more general than identifie since there are many kinds of expressions that could be used to name boxes. \ mentioned that p^ could be the name of an object pointed to by pointer p, a other names are complex expressions such as p^.f[2]. The latter expression ref to the array element number 2 of the field f of the record pointed to by pointer

---

linked list defined in Example 1.1, where Fig. 1.3 is a typical linked list terminat by a NIL pointer. Technically speaking, objects of type CELL of that example cou also loop back on themselves, as suggested by Fig. 1.13, but we tend not to use su structures in practice. ✦

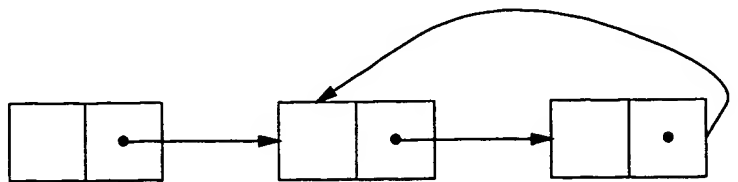Fig. 1.13. Linked list not terminated in NIL.

Procedures and Functions

Procedures also have associated types, even though we do not associate boxes "values" with procedures, as we do with program variables. For any list of typ $T_1, T_2, \ldots, T_n$, we can define a procedure with n parameters consisting of th types, in order. This list of types is the "type" of the procedure. A procedure m